**LA-UR** -90-2004

CONF-900621?--1

LA-UR--90-2004

DE90 013173

TITLE CROSS-VALIDATION, LEARNING SET TRANSFORMATIONS, AND GENERALIZATION

AUTHOR(S) David H. Wolpert

SUBMITTED TO Rocky Mountain Conference on Artificial Life, Las Cruces, NM, June 1990

**MASTER**

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

# CROSS-VALIDATION, LEARNING SET TRANSFORMATIONS, AND GENERALIZATION

by David H. Wolpert
Theoretical Division, MS B213, LANL, Los Alamos, NM, 87545 (dhw@coot.lanl.gov), (505) 665-3707.

Abstract: This paper discusses using cross-validation as an aid to generalization. It starts by showing how to use cross-validation to decide amongst a set of generalizers when the learning set consists of examples of the text-to-phoneme problem. In addition to reproducing the learning set perfectly, the generalizer so chosen by cross-validation has an error rate on the *testing* set (7%) close to that which NETtalk has on the *learning* set (5%). This paper then presents an example of using cross-validation as part of a front-end to generalizers, i.e., as part of an algorithm for pre-processing a learning set of input/output examples before trying to generalize from it. The results of thirty-six comparisons between the performance of a generalizer and the performance of the generalizer with this front-end are presented. These comparisons involve numerical, Boolean, and visual tasks. In all but one of the comparisons the front-end improves the generalization performance, often inducing perfect generalization. (The average ratio of the generalization error rate with the front-end to the generalization error rate without the front-end is .23, +/- .05.) Finally, this paper ends by discussing some of the subtler mathematical issues involved in using cross-validation to help generalization.

## INTRODUCTION

This paper concerns the problem of inferring a function from a subset of $\mathbf{R}^n$ to a subset of $\mathbf{R}$ (the *parent* function) given a set of m samples of that function (the *learning set*). The subset of $\mathbf{R}^n$ is the *input space*, and the subset of $\mathbf{R}$ is the *output space*. A *question* is any input space value not contained in the learning set. An algorithm which makes a guess as to a parent function (for any value of n), basing the guess only on a learning set of m $\mathbf{R}^{n+1}$ vectors read off of that parent function, is called a *generalizer*. A generalizer infers an appropriate output for a question via the parent function it guesses.

Much of the research on generalization has followed two approaches. The first is to "just do it": build an engine which reproduces the learning set, and then use that engine to generalize. An example of this approach is back-propagated neural nets [1]. Schemes of this type devote most of their attention to reproducing the learning set and expend little (if any) effort at trying to ensure good generalization. In effect, these schemes simply *hope* that good generalization accompanies reproducing the learning set. The advantage of generalizers of this type is their wide range of applicability.

The second approach to generalization research is typified by Valiant-style machine learning [2]. Here complete rigor is insisted upon (i.e., no reliance on hope), but generality and real-world applicability are sacrificed to achieve the rigor. For example, most such schemes can not even address the following toy problem: "You have a learning set consisting of 3 input/output pairs where the input space is one-dimensional, (1.0, 1.3), (2.0, $\pi$), and ($\sqrt{2}$, 5.0), and no other knowledge; guess the output corresponding to 5.5". Such schemes don't even address the full problem of generalization, as it was defined above.

There are a number of approaches which fall in between these first two, using as much rigor as possible, but also exploiting heuristics, to maintain applicability to real-world problems. Amongst these approaches are (roughly in increasing order of sophistication) Holland's classifier system [3], memory-based reasoning schemes [4], regularization theory [5], other schemes for overt surface fitting of a parent function to the learning set [6-10], and information-theory based approaches [11, 12]. This paper discusses another intermediate approach, cross-validation [13, 14]. Cross-validation is simply the idea that if you have a set of generalizers and a learning set and have to decide which of those generalizers to use with that learning set, you should choose the generalizer which performs best at guessing one part of the learning set after being taught with the rest of the learning set. The performance is usually measured with a squared guessing error, averaged over (some of) the partitions of the learning set into two subsets. Many of the information-theory based approaches to generalization can be viewed as variations of cross-validation (see footnote 4 in [14]).

Section I of this paper is an example of using cross-validation for the text-to-phoneme problem. This example illustrates the ease with which cross-validation can result in generalization superior to that of back-propagation. Section II presents a more sophisticated example of using cross-validation, in this case using a so-called "lattice generalizer front end" to pre-process a learning set before sending it on to a generalizer. Section III presents tests of the efficacy of this front end. The conclusion from this example is that cross-validation is a very powerful aid to such pre-processing. Finally, section IV is a cursory overview of some of the subtler mathematical issues involved with using cross-validation.

## I. USING CROSS-VALIDATION TO BEAT NETtalk

NETtalk is a feedforward neural net constructed by Sejnowski and Rosenberg via back-propagation for the task of reading English text aloud [15]. The input space consists of (an encoding of) seven letters. The output of the parent function is a multi-dimensional representation of the phoneme an English speaker would assign to the middle (i.e., fourth) of the seven letters if the reader encountered them while reading aloud. NETtalk has an error rate of ~ 5% on its learning set, and ~ 22% on its testing set.

As a crude test of its utility, cross-validation has been used to build a generalizer for the reading aloud task and the resulting performance has been compared to that of NETtalk. This generalizer, using the same learning and testing sets as Sejnowski and Rosenberg say they used [16], has an error rate on the learning set of 0% and an error rate on the testing set of 7%, less than one third the error rate of NETtalk.

The details of how this generalizer is built can be found in [7]. One starts with a weighted average generalizer, one of the simplest kinds of generalizers. For this problem the weighted average generalizer works by finding the four elements of the learning set which are closest to the question, and taking a weighted average of their outputs to make the guess. The weighting factor is the reciprocal of the distance between the question and the learning set element, so the nearer an element of the learning set is to the question, the more strongly it is weighted in making the guess. Using a simple Hamming metric, such a generalizer results in 0% error on the learning set and 18% error on the testing set.

To use cross-validation, we can consider a set of weighted-average generalizers, all of which differ from one another by using different versions of the standard Hamming metric. Specifically, we can consider generalizers of the form

$$\text{guess} = \left(\sum_{i=1}^{4} [y(\mathbf{x}_i) / d(\mathbf{q}, \mathbf{x}_i)]\right) \Big/ \left(\sum_{i=1}^{4} [1 / d(\mathbf{q}, \mathbf{x}_i)]\right), \text{ where } d(\mathbf{a}, \mathbf{b}) = \left(\sum_{\alpha=1}^{7} [\rho_\alpha (1 - \delta(a_\alpha, b_\alpha))]\right), \delta(.,.)$$

is the Kronecker delta, letters in bold are vectors, roman subscripts of vectors index the (input vectors of the) elements of the learning set nearest in input space to the question **q**, greek subscripts run over the input components of a particular vector, and the $\rho_\alpha$ are a set of seven real valued constants. The usual Hamming metric has all the $\rho_\alpha = 1$.

Different sets of $\rho_\alpha$ give different guesses for the same learning set and question, i.e., they give different generalizers. Therefore we can vary the $\rho_\alpha$, measure the resultant cross-validation error for the learning set, and choose the $\rho_\alpha$ with the lowest such error. We then use the generalizer induced by this optimal set of $\rho_\alpha$ to generalize from the entire learning set. The results of this strategy are depicted in figure 1. Using the generalizer with minimal cross-validation error, we would choose the generalizer corresponding to the leftmost circle in the figure. This happens to be the generalizer which also gives the lowest error rate on the testing set. In fact, this generalizer has an error rate of only 7% on the testing set; cross-validation has greatly improved generalization efficacy.

## II. MOTIVATION AND DEFINITION OF LATTICE GENERALIZERS

A generalizer is *local* if the guess it makes in response to a question is determined by a proper subset of the elements of the learning set. Usually this subset consists of elements which are near (in input space) to the question. Such generalizers therefore rely on a *fan* of vectors connecting a question to those elements of the learning set which are deemed important (for the task of making a guess for the question's output). Weighted-average generalizers, memory-based reasoners [4], the local linear technique [9] and HERBIEs [6, 7] are all local generalizers. Back-propagated neural nets are somewhat local - although not explicitly local generalizers, their guessing is often far more dependent on the nearby elements of the learning set than on any others. A generalizer is *global* if its guess is determined by all elements of the learning set. An example of a global generalizer is fitting the elements of the learning set exactly with a linear combination of elements from a set of basis functions. Global generalizers have the advantage that they use all of the information in the learning set. Local generalizers have the advantage that they don't "overfit" the data of the learning set, and therefore tend to be robust.

*Lattice generalizers* (LGs) are a means of incorporating global information into a local generalizer so that all the information in the learning set is exploited (while robustness of generalization isn't sacrificed). A full description and discussion of LGs can be found in reference [17]. Here we are concerned with input-independent LGs. In their simplest form, such LGs assume that the question and the elements of the learning set all live on a lattice in the input space (hence the name lattice generalizer). The researcher provides a local generalizer for such a question and learning set. Then the LG takes the original learning set and question and transforms them into a "reduced" learning set and "reduced" question, both of which live in the same "reduced" input/output space. The algorithm for accomplishing the reduction is known as the *Lattice Generalizer Front End* (LGFE). In LGs, instead of feeding the original learning set and the original question straight to the original (so-called "back-end") local generalizer, the reduced learning set and reduced question are fed to the original generalizer instead. The LGFE together with the back-end generalizer constitutes the full LG.

It is through the LGFE that lattice generalizers make use of global information. The LGFE constructs the reduced learning set from the output components of the original learning set. The input components of the original learning set serve only to decide how to transform the output components of the original learning set into the reduced learning set. This decision is made using cross-validation, in concert with the back-end generalizer. A loose description of how an input-independent LGFE works follows:

Assume we have a fan consisting of m input space vectors $\{r_i\}$, $1 \le i \le m$, which connects a question to m elements of the learning set. Assume further that this fan connects some of the elements of the learning set to each other (i.e., assume there exist elements of the learning set such that if they are taken as the base for the fan, the tips of the fan also lie in the learning set). For any such base in the learning set, since the tips of the fan lie on elements of the learning set, we can examine the set of m outputs of the learning set at these tips. These m outputs of the learning set are taken as the specification of a single point in an m-dimensional input space. The output associated with this m-dimensional input point is the output of the learning set element at the base of the fan (see figure 2). By finding all bases for the fan which lie in the learning set we can build a set of pairs, {m-dimensional input vectors, associated output}. In this way we have used the fan to build a new (reduced) learning set from the output components of the old one. We can use the fan to read off the reduced question as well, and then we can feed this reduced question along with the reduced learning set to the original back-end generalizer (instead of feeding it the original learning set).

If we go through this procedure for all fans which connect together elements of the original learning set, then we will build a set of reduced learning sets, any one of which can be fed to the back-end generalizer. To decide which of the fans to use, we apply cross-validation: pick the fan such that the new learning set it produces has lowest possible average error at guessing part of itself from the rest of itself. (This error is determined by running the reduced learning set through the back-end generalizer.) After choosing a fan in this way and using it to build a reduced learning set and reduced question, use the back-end generalizer with that reduced learning set and question to make the guess.

## II. TESTS OF LATTICE GENERALIZERS

The results of thirty-six tests of the behavior of LGs for nine separate problems are presented in figures 3(a) through 3(i). The tests involved running the LGs on testing sets entirely different from the learning sets with which they were taught. All of the experiments were done with fans having only one tip.

To make things difficult, all of the experiments took place on a hypercube and used explicit surface-fitter generalizers as the back-end (see [4-6] and [9]). Such generalizers usually perform much better when the data isn't binary-encoded; such encoding constitutes handicapping the generalizers severely. As a result, these are good tests of the efficacy of feeding a generalizer with an LGFE.

For all of the nine problems presented here, the input space is six-dimensional, and each component of an input vector can be either a 1 or a 0. Therefore there are 64 possible input vectors; the sum of the cardinalities of the learning set and the testing set is always 64. The figures represent the average behavior of an LG for the questions in the testing set. Four testing set cardinalities were investigated for each of the nine problems; 8 elements, 16 elements, 32 elements, and 48 elements (corresponding to learning sets of 56 elements, 48 elements, etc.). For each size testing set, 20 tests were run by creating a random testing set of the given size. The figures give the average percentage of wrong guesses over the 20 tests, for each of the nine problems, for all four testing set cardinalities. This gives a total of 36 tests.

Each figure shows both the generalization error rate of the back-end used without an LGFE and the error with the LGFE in place. Each figure also shows the error rates of optimal random guessing (i.e., of always guessing the mode of the output distribution) and of back-propagation. (The back-propagation data was generated with the George Mason University BPS 1.01.). The back-propagation wasn't fed via an LGFE - it was trained with the original learning set. Since back-propagated neural nets are *designed* for the hypercube, they were used here as a sort of crude benchmark for problems taking place on a hypercube. In general, like any other generalizer, the generalizing behavior of back-propagation should usually be improved if it is fed through an LGFE.

Tests 1-4: Figures 3(a) through 3(d) depict an LG's behavior for four binary-encoded numerical problems. The parent function for each problem has a 6 dimensional input space. The first 3 dimensions are the binary encoding of a number between 0 and 7, as are the last 3 dimensions. The output is one of the bits of the binary encoding of the sum of these 2 numbers. Figure 3(a) represents the results for the first (least significant) output bit, figure 3(b) the second output bit, and so on through figure 3(d). For example, with input (1, 1, 0, 0, 1, 0), the four output bits should be the binary encoding of 6 + 2 = 8. So for the situation depicted in figure 3(a), the correct output of a generalizer would be a 0, as it would for figures 3(b) and 3(c); figure 3(d) would have a 1 as the correct output.

The back-end generalizer used for these first four tests is a weighted average generalizer running over the seven elements of the learning set nearest to the question. As with all the other tests presented here, for these tests the dark circles lying on the 0% axis have exactly 0% error in guessing. For example, in figure 3(a), for the case of 32 element learning sets there were zero generalizing errors out of 20 × 32 = 640 trials. The LGFE induced *perfect generalization!*

Test 5: Figure 3(e) also represents a test of a binary system with 6 dimensions of input and therefore with 64 possible inputs. Here the parent function has a boolean nature: if the first dimension has value 1, the output is the value of dimension 3, otherwise it's the value of dimension 6. Note that three of the input dimensions are unimportant; they're red herrings. The generalizer is again a (normalized) weighted average generalizer.

Test 6: Figure 3(f) again depicts a situation where the input space is binary and 6-dimensional, but now the output space is no longer binary. The problem here is of a visual nature: taking the 6 input dimensions as a 6-pixel wide window, scan from the left of the window until you hit a 1; the output is the number of scanned pixels. Here the generalizer fed by the LGFE works by finding the 7 points of the learning set nearest the question, fitting a hyperplane to them, and outputting the height of the hyperplane when the input is the question. (If the 7 nearest points are linearly degenerate, weighted averaging is used instead.)

Tests 7 and 8: Figures 3(g) and 3(h) depict the results of a test related to the (two clump)/( three clump) problem: count the number of 1's in the 6 components of the input. Test 7 (figure 3(g)) uses a hyperplane fitter as its generalizer, just like test 6, whereas test 8 uses a weighted average generalizer. For the counting problem, the parent function is given by output = $x_1 + x_2 + x_3 + x_4 + x_5 + x_6$, the sum of the 6 components of the input. Since this function is a hyperplane, the hyperplane fitter generalizes better than the weighted average fitter. (The errors of the hyperplane fitter occur when the nearest neighbors of the question are linearly degenerate, so some algorithm different from fitting a hyperplane has to be used). Even so, the generalizer consisting of an LGFE feeding into a hyperplane generalizer performs better than the hyperplane generalizer alone; it generalizes perfectly in fact. Just as a hyperplane fitter is well-suited to guessing this parent function, a weighted average generalizer is poorly suited to this task. Nonetheless, except for the case of 8 point testing sets, feeding the output of the LGFE into the average weighted generalizer still produces better generalization than running an average weighted generalizer without the LGFE front-end.

Test 9: Figure 3(i) depicts the results for the parity problem. The problem has 6 bits of input as usual, with 1 bit of output. The output is a 1 if there is an odd number of 1's in the input, 0 otherwise. (Note this function is just the output of the count function evaluated mod 2, i.e. it's the low bit of the count problem's output if that output is binary-encoded.) The back-end generalizer is the hyperplane fitter. Note how poorly both back-propagation and the hyperplane fitter perform. The reason for this isn't hard to find: both back-propagation and the hyperplane-fitter are local algorithms. But on a hypercube the six nearest neighbors of any question will differ from that question by one bit. This means that for the parity problem the outputs of the six nearest neighbors is exactly opposite to the correct guess. As a result, local generalizers perform extremely poorly.

The parity problem is a good example of how pernicious binary encodings of problems are, taking place as they do at the vertices of a hypercube. After all, the hyperplane fitter does much better at guessing the entire output of the count problem than at guessing the low bit of that output (given in the parity problem). Note, however, that the full LG handles both the parity problem and the full count problem quite well, hypercube or no. It handles them perfectly, in fact.

The results of these tests are unambiguous. In all but one of the thirty-six cases presented here, using the LGFE front-end produces better generalization. (The only case where there was no improvement had the LG performing just barely worse than the straight back-end.) Often use of the LGFE even results in perfect generalization. The average value of the ratio of the generalization error rate with the LGFE to the error rate without one is .23, with an estimated error in this mean of .05.

In addition to these empirical results, there are many theoretical advantages to LGs as well. A list of some of them can be found in [17].

## IV. SELF-GUESSING AND CROSS-VALIDATION

As presented so far in this paper, cross-validation is used as a means for choosing amongst a fixed finite set of possible generalizers of a learning set. It can also be used to try to *construct* the optimal generalizer of a given learning set from first principles. The idea is to demand perfect cross-validation on the learning set along with some other restrictions, and thereby deduce the (hopefully unique) optimal generalization of the learning set. When used in this way cross-validation is referred to as "self-guessing" [14]. An overview of some of the more interesting aspects of self-guessing is presented in this section.

There are several different versions of self-guessing. For example, if we only demand that the generalizer perfectly guess a subset of the learning set when taught with the (sufficiently large) remainder of the learning set, then we are enforcing *weak* self-guessing. We can instead demand, somewhat in the spirit of distribution-free learning [2], that the function f(.) from inputs to outputs guessed by the generalizer have the following property: f(.) reproduces our learning set, and if we were given any learning set of samples of f(.), then the generalizer would guess the input-output function f(.). This is called *strong* self-guessing; strong self-guessing implies weak self-guessing, but not visa-versa.

There are a number of interesting facts about self-guessing. For example, there are an uncountably infinite number of self-guessing generalizers (both weak and strong) for any learning set. On the other hand, no generalizer is perfectly self-guessing (either strongly or weakly) for every learning set. Moreover, there are learning sets for which there are no self-guessing generalizers whatsoever, if we also require that the generalizer obey the invariances of Euclidean space.

Of paramount importance is the following fact: It might be hoped that we could construct a set of restrictions on the set of generalizers such that, for any particular learning set, there is only a single generalizer which both meets the restriction and is self-guessing (either weakly or strongly) for that learning set. Unfortunately, it can be proven that there is no such set of restrictions on the set of generalizers. Any such set of restrictions will either be under-restrictive (i.e., there are learning sets for which there are more than one self-guessing generalizer which meet the set of restrictions) or over-restrictive (i.e., there are learning sets for which there are no self-guessing generalizers which meet the set of restrictions).

This means that there are three ways to exploit self-guessing. The first is to still demand perfect self-guessing, but not to demand that a particular set of restrictions is met exactly. Instead, one chooses the self-guessing generalizer which obeys most closely (according to an appropriate metric) the set of restrictions. The second approach is to demand neither perfect self-guessing nor that any set of restrictions is met exactly, and instead choose that generalizer which is as close as possible to being both self-guessing and in accord with the set of restrictions. Finally, one can demand that some set of restrictions is met exactly, and then choose the generalizer which is most closely self-guessing. The set of restrictions in this case is often implicit. (For example, in section I, the set of restrictions was that the generalizer be expressible as a weighted-average generalizer.) When used with weak self-guessing, this last approach amounts to standard cross-validation.

## CONCLUSION

This paper has presented experimental evidence that cross-validation aids generalization greatly (i.e., results in markedly improved guessing). The evidence concerns both naive uses of cross-validation and using it in a more sophisticated manner, as a learning set preprocessor. In addition, this paper has surveyed some of the theoretical aspects of cross-validation and discussed their implications for how and when cross-validation can be profitably employed.

## REFERENCES

[1] Rumelhart, D. E., and McClelland, J. L., Explorations in the microstructure of cognition, volumes I and II. MIT Press, Cambridge, MA, 1986.

[2] Valiant, L., A theory of the Learnable. *Communications of the ACM*, **27**, 1134-1142, 1984.

[3] Holland, J., Adaptation in natural and artificial systems, University of Michigan Press, 1975.

[4] Stanfill, C., and Waltz, D., Toward memory-based reasoning, *Communications of the ACM*, **29**, 1213-1228, 1986.

[5] Poggio, T., and staff, MIT AI Lab, MIT progress in understanding images. To be published in L. Bauman (Ed.), *Proceedings of the image understanding workshop*. McLean, VA, 1988.

[6] Wolpert, D., A benchmark for how well neural nets generalize, *Biological Cybernetics*, **61**, 303-313, 1989.

[7] Wolpert, D., Constructing a generalizer superior to NETtalk via a mathematical theory of generalization. To appear in *Neural Networks*, July 1990.

[8] Wolpert, D., A mathematical theory of generalization: part I. *Complex Systems*, in press.

[9] Farmer, J.D., and Sidorowich, J.J., Exploiting chaos to predict the future and reduce noise, Los Alamos report LA-UR-88-901, 1988.

[10] Omohundro, S., Efficient algorithms with neural network behavior, *Complex Systems*, **1**, 273-347, 1987.

[11] Akaike, H., Information theory and an extension of the maximum likelihood principle, *IEEE Trans. Automatic Control*, **AC-19**, no. 6, 716-723, 1974.

[12] Rissanen, J., Stochastic complexity and modeling, *The Annals of Statistics*, **14**, 1080-1100, 1986.

[13] Efron, B., Computers and the theory of statistics: thinking the unthinkable, *SIAM REVIEW*, **21**, 460-480, 1979.

[14] Wolpert, D., A mathematical theory of generalization: part II. *Complex Systems*, in press. "Cross validation" is a special case of the property of "self-guessing" described in this paper.
88.

[15] Sejnowski, T.J., and Rosenberg, C.R., NETtalk: a parallel network that learns to read aloud. Johns Hopkins University Electrical Engineering and Computer Science technical report JHU/EECS-86/01, 1988.

[16] Carterette, E.C., and Jones, M.H., Informal Speech, UCAL Press, Los Angeles, 1974.

[17] Wolpert, D., A new technique for improving the performance of any generalizer, Los Alamos report LA-UR-90-401. Submitted to *IEEE PAMI*.

•

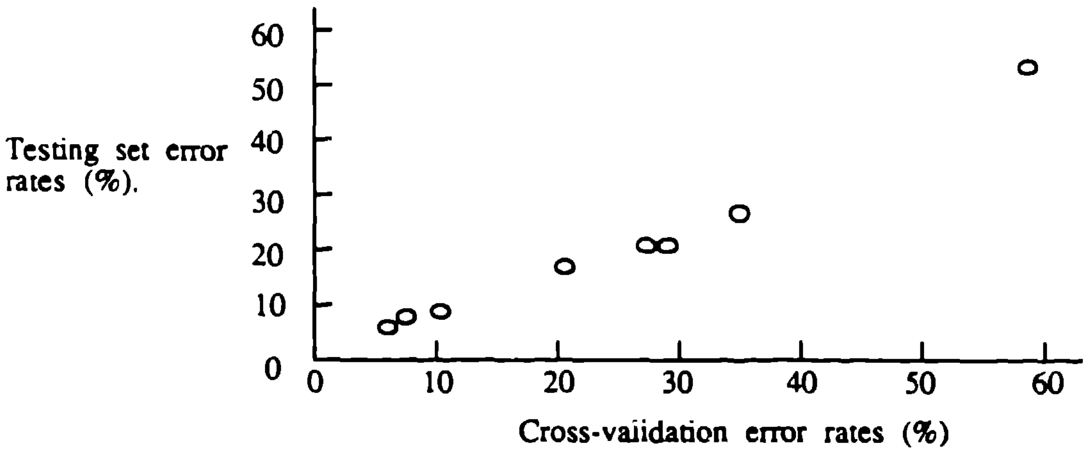
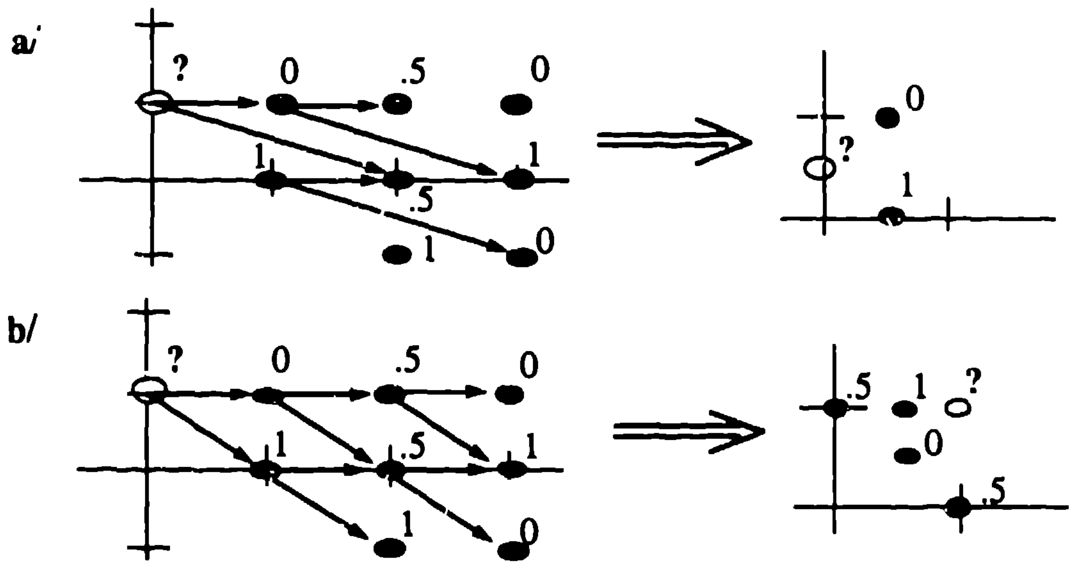
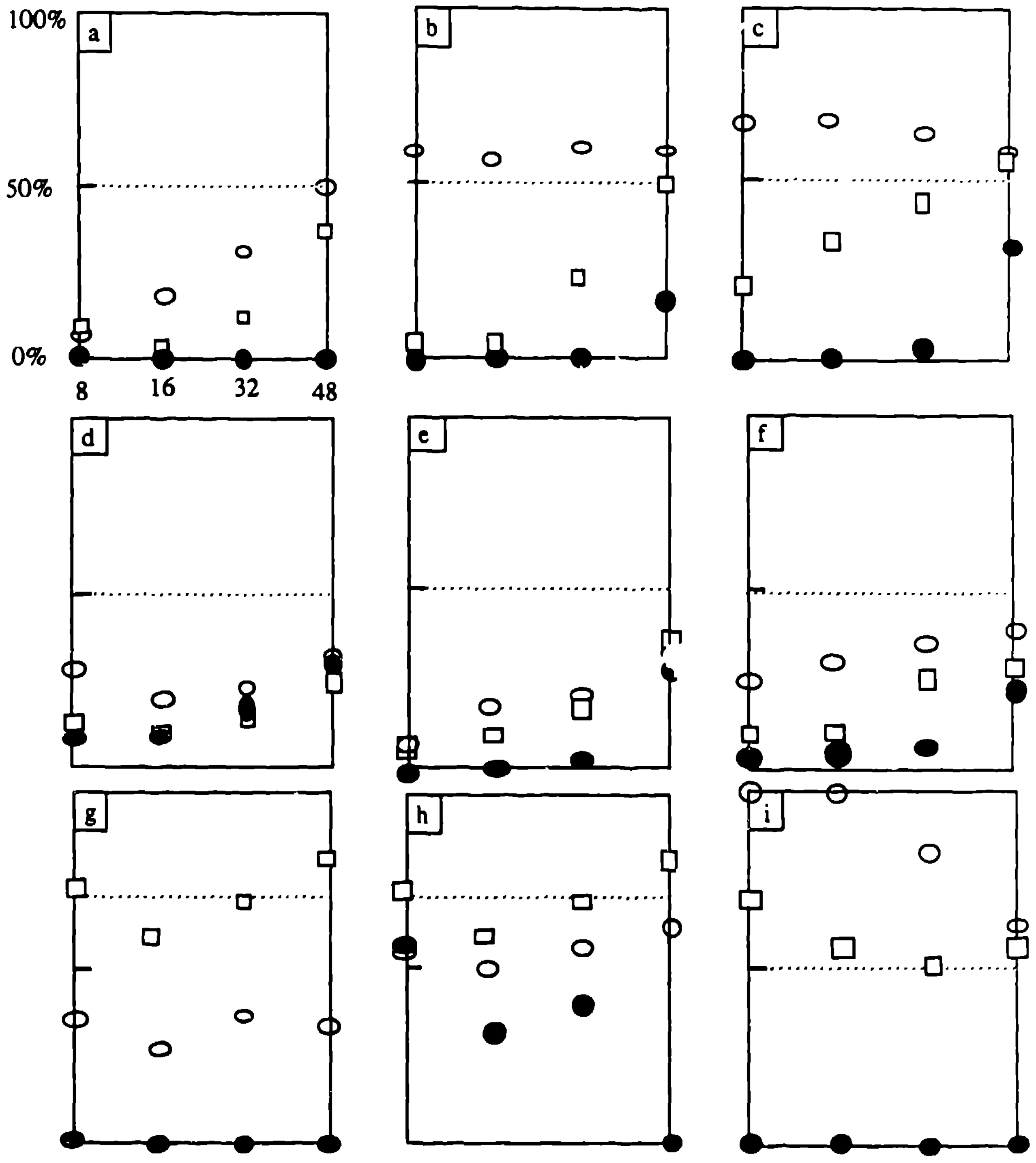


Figure 1. The horizontal axis gives the error rate of eight different generalizers for guessing 2189 elements of the (reading aloud) learning set when taught with the rest of the learning set. The vertical axis gives the error rates for guessing the 2189 elements of the testing set when taught with the full learning set (see reference [7] for details). The correlation between the two errors is clear. The cross-validation error rate of NETtalk is unknown, but its error rate on the testing set is 22%.



Figures 2a and 2b. An input-independent LG. The left half of each figure represents a learning set and a question. The elements of the learning set are solid circles, the question is an open circle. There are two dimensions of input. The output value of each element of the learning set is indicated; here the outputs are all either 0, .5, or 1. Also shown in the left half of each figure is a 2-tip fan, translated over several different bases. The fan in (1a) makes a connection amongst the elements of the learning set twice. The fan in (1b) makes a connection amongst the elements of the learning set four times. Every connection defines an element in a new learning set over a 2-dimensional input space. In (1a) for example the two such elements are ((.5, 1), 0) and ((.5, 0), 1). The new learning sets made in this way are indicated in the right halves of each figure. The back-end generalizer uses these new learning sets. The LGFE served to create these new learning sets. To answer the question in (1a), the question is converted (via the fan of (1a)) to the new question (0, .5). The back-end generalizer then takes this new question and the new learning set and comes up with a guess. To decide which of the fans to use, we find the fan



Figures 3(a) through 3(i). Open circles represent the guessing of a generalizer. Solid circles represent the guessing of that generalizer being fed by an LGFE. Squares represent the guessing of back-propagation. The dotted line indicates the behavior of guessing the mode of the original learning set's output distribution.The horizontal axis gives the testing set size, and the vertical axis gives the error rate at guessing the elements of the testing set. The specific problem represented by each of the 9 figures is described in the text.